LULEÅ UNIVERSITY OF TECHNOLOGY

2001:65 HIP

BACHELOR'S THESIS

Service Location Protocol

Maria Norberg and Catarina Burman

BACHELOR OF SCIENCE PROGRAMME
COMPUTER ENGINEERING

Skellefteå Campus

2001:65 HIP · ISSN: 1404 - 5494 · ISRN:LTU - HIP - EX -- 01/65 --SE

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

Abstract

Everybody who works within a network will sometimes be of need to access a special type of service (for example a printer). When this occur can a protocol like the Service Location Protocol be of help, so the user just need to know what kind of service he/she whishes for.

Ericsson Erisoft has a problem solving "loose connections". The fast technical development today demands the capability to easily replace different components. It is impossible to know what services that are needed in the near future. A protocol that maybe can solve this is the Service Location Protocol (SLP).

Our assignment where to investigate the protocol capacity and develop a prototype to demonstrate the technology. Another part of the job where to search after suitable tools that can make the introduction of SLP easier.

We found out that the Service Location Protocol could solve this problem. The only drawback is that SLP needs more testing and is not a standard yet. With help from the tools that we found it became rather easy to develop a prototype but something is unstable with either the tools or our program.

Ericsson Erisoft 9/S94:002 Rev E

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

Acknowledgment

First we would like to thank Ericsson Erisoft for giving us the opportunity to fulfill our education. Our instructor Ulrik Forsgren is also worth a big thank you note for all the help that he has given us, and all the questions that he has answered. We would also thank all the other personnel at N/N for making our time here pleasant.
Last but not least we would like to thank the teachers and studentexp. personnel at "Luleå tekniska universitet Instutionen i Skellefteå".

Catarina Burman and Maria Norberg - July 4, 2001

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

Contents

1 A List of Abbreviations Used in this Document 5
2 Introduction 6
2.1 Background 6
2.2 Purpose 6
2.3 Main Task 7
3 SLP 8
3.1 What is SLP 8
3.1.1 The Concept for SLP 9
3.2 Services 10
3.2.1 Service Agent Server 10
3.2.2 Service Advertisement 10
3.2.3 Service Type Templates 10
3.2.4 Service URL 11
3.2.5 Service Attributes 11
4 How SLP Works 12
4.1 SLP API 12
4.2 Messages 13
4.3 SLP Architecture 14
4.4 Security 15
5 NMS 17
5.1 Network Supervision 17
5.2 Workflow Support 17
5.3 Equipment Handling 17
5.4 An Example of NMS in a Mobile Network 18
6 Integration of SLP into NMS 19
6.1 Where do We Integrate SLP into NMS? 20
6.1.1 CDS 20
6.1.2 RANOS 20
7 Different Tools for Developing SLP 21
7.1 OpenSLP Version 0.9.0 21
7.1.1 When is Slpd Needed? 22
7.1.2 When is Slpd not Needed? 22
7.1.3 The slp.conf File 22
7.1.4 The Static Registration File 23
7.1.5 Security 23
7.2 Solaris 8 slpd 23
7.2.1 How SLP is Implemented 23
7.2.2 Monitoring the SLP Traffic 24
7.3 Mini SLP 25
8 Prototype Developing 26
8.1 Prototype Requirements 26

8.2 Risks in the Project 26
8.2.1 Environment 26
8.2.2 SLP 27
8.3 Final Prototype Design 28
8.3.1 The Program Written in Java 28
8.3.2 The Program Written in C++ 29
8.4 SLP Technology Test Programs 30
8.4.1 Implementation Environment Research 30
8.4.2 Developed Programs 31
8.4.2.1 SLP Implementing Test 31
8.4.2.2 Prototype in C++ 31
8.4.2.3 Prototype in Java 2 32
8.4.2.4 The Final Prototype 32
8.5 Prototype Tests 33
8.5.1 Configuration 33
8.5.2 Registration 33
8.5.3 Services 34
8.5.4 Errors 34
8.5.5 Networks 34
8.6 The Final System Design 35
9 Conclusions 36
9.1 Tools 36
9.1.1 Functionality 36
9.1.2 MSLP 36
9.1.3 OpenSLP 36
9.1.4 SLP in Solaris 8 37
9.1.5 Reasons for Chosen Different Tools 37
9.2 Our Application 38
9.3 Multicast and Firewalls 38
9.4 Predicted Risks 39
9.5 Final Conclusions 39
9.5.1 SLP in the Future 41
10 Glossary 42
11 References 47
Appendix A – Configuration of DA and SA 49
Appendix B – Example of a register file 59
Appendix C – Example of a RANOS file 60
Appendix D – Special information to the SLP tools 61

1 A List of Abbreviations Used in this Document

AM	Access Module
API	Application Programmers Interface
CDS	Common Directory Server
DA	Directory Agent
DHCP	Dynamic Host Configuration Protocol
FM	Function Module
FullS	Full Servicer
IEFT	Internet Engineering Task Force
IP	Internet Protocol
MSC	Mobile Switching Center
MSLPd	Mini Service Location Protocol daemon
NMS	Network Management Solution
OpenSLP	The open source Service Location Protocol project
PM	Presentation Module
PKI	Public Key Infrastructure
RANOS	Radio Access Network Operation Support
RBS	Radio Base Stations
RFC	Request for Comments
RNC	Radio Network Controller
SA	Service Agent
SLP	Service Location Protocol
Slpd:	Service Location Protocol daemon
SMS:	Short Message Services
SPI:	Security Parameter Index
SRVLOC WG:	Service Location Working Group
TCP	Transmission Control Protocol
URL:	Uniform Resource Locator
UA	User Agent
UDP	User Datagram Protocol

Ericsson Erisoft 9/S94:002 Rev E

2 Introduction

2.1 Background

NMS is a network management system for mobile network. The system supervises a mobile operator's whole net with a huge number of equipment. The equipment include everything from Base Stations, Base Station Controllers, MSC (Mobile Switching Centre), net transport equipments, SMS (Short Messages Services) etc. In its turn one network management system consists of several different applications in a distributed system. Examples of applications can be Fault Management (FM), Performance Management (PM), Trouble Ticket, Inventory etc. These applications are their own products that are integrated in NMS. The goal is to integrate in "Best Of Breed" products. The choice of applications however, must be able to suite depending on that the mobile operator maybe already has an application like PM that he/she want to integrate into the system. This means that NMS must have an architecture that makes it possible to change the component parts (applications) in an easy way. Another aspect is when an application upgrade is done, then it would be desirable that the upgrade did not influence the integration.

Ericsson Erisoft needs a way to solve these "loose connections" in their integrations.

2.2 Purpose

The purpose of this document is to make a textual presentation of our bachelor thesis. This completes our education in computer engineering at the institution in Skellefteå of Luleå Technical University.

Ericsson Erisoft 9/S94:002 Rev E

2.3 Main Task

The purpose of the thesis is to examine if the technology with Service Location Protocol (SLP) can be one way to accomplish "loose connections". Can this be a way to make the integration easier in systems like NMS?

The work consists of the following parts:

1) A study, whose purpose is to learn about SLP. Make estimation if it can be a technology that makes the integration of component parts in a NMS, map on architecture, release handling and choice of different manufacture in component parts easier.

2) Developing of a simple prototype to test the technology.

3) Search on the Internet if there are any components and tools that can accomplish easier introduction of SLP.

Ericsson Erisoft 9/S94:002 Rev E

3 SLP

3.1 What is SLP

SLP (Service Location Protocol) is a standard track protocol proposed in the IEFT by the SVRLOC working group. The SVRLOC WG was formed in 1993 and completed in 2001. The protocol has two versions but we have concentrate on version 2.The protocol provides a framework to allow networking applications to discover the existing, location and configuration of networked services in enterprise networks. It is a way for clients to dynamically discover network services and is designed to give a simplified discovery and use of a network, such as writers, Webb servers, fax machines, files systems and backup units. It is specially adapted for client-server applications and to open communication between nodes in a network that offers or uses common services. Service Location Protocol version 2 is only a proposed standard. This means that it is not tested enough to be a standard and should therefor be threaten as an immature specification that can be changed. To implement a proposed standard like this into a disruption-sensitive environment is not recommended.

SLP can eliminate the need for users to know the name or location of the network hosts. When using the protocol it is only necessary for the users to know the description of the service he/she is interested in. It is based on that description that the SLP can return the URL of the desired service.

It is an unicast and a multicast protocol. This means that messages can be sent to one agent at a time (unicast) or to all agents that are listening at the same time (multicast).

Multicast is sent to all the nodes that have joined a multicast group in the network and is forwarded by the routers. Broadcast can be chosen and is then sent to all the nodes in a network and the routers do not forward it. Broadcast should be avoided because of the uncontrolled sending to all the nodes.

Ericsson Erisoft 9/S94:002 Rev E

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

3.1.1 The Concept for SLP

SLP consists of many processes.
- User Agent
- Service Agent
- Directory Agent

In this protocol is an agent a software unit that takes care of the SLP message.

User Agent (UA) is a software unit that looks after where one or many services are located. It is a process for where the services are located or a library that allows SLP request from a client process. UA automatically forward to a DA if a DA exist, otherwise the request goes to a SA.

Service Agent (SA) is a software unit that supplies the location of one or many services. This agent keeps a description of where the service is located or a library that allows software services to register and deregister themselves with help of the SLP framework. SA answers the UA request for a service demand and then discovers DAs and register "Service Advertisement" at the DA.

Directory Agent (DA) is a software unit that acts as a centralized storage for information about where the service is located. It collects and keeps information about SAs. DA answers only on a UA request if it has information about the services that the UA asks for.

The different processes use scopes.

Scope is an administrative or logical grouping of services. It is a string that is being used to control the availability for services that are offered. Each SLP agent is configured with one or more scope strings. DA stores only services that are offered and use a scope string that matches the scope that they are configured with. DA and SA must be configured with scopes because they shall for example be able to answer a DA or SA query with a list over the scopes that they support.

Ericsson Erisoft 9/S94:002 Rev E

3.2 Services

3.2.1 Service Agent Server

Many operating systems platforms allow only one process at a time to listen at a special port. SA is forced to listen to a multicast query address (in SLP V2 239.255.255.253) for protocol queries about services. When implementing SLP framework on such a platform, one process must listen while the SA communicates trough another mechanism. The listening process is called SA server.

3.2.2 Service Advertisement

Service advertisement is an URL, sometimes together with the service special features. This becomes available for the UA either directly by the SA or through the DA. Service Advertisement characterizes a service authority through a collection of attributes that describes the service. One of the distinguishing features is Service Type that describes a class of services that share the same hallmark. Service Advertisement also contains another important information; namely how to reach services (Access Point). This is information enough for the UA to get hold of the service. An Access Point contains machine hostname or IP address and possible a port number. Some services demands more information so that one can reach them, this information can also be included.

3.2.3 Service Type Templates

A document that describes the URL syntax for a special type of services and a definition for all the service attribute, default and restrictions about what values the different attributes can have included. Examples for how to write different URLs and attributes can be found in RFC 2609.

Ericsson Erisoft 9/S94:002 Rev E

3.2.4 Service URL

A service of a special kind that offers its availability and what type of service it can offer. URL includes the Service Access Point (hostname or IP address and possible a port number) and optional base configuration parameter that shows the syntax for the service.
An example of how the service URL can be written:
service:BasicCM:icpname://<ip>:<ev port>/com/ericsson/units.
The abstract service type is "service:BasicCM" meaning that it could be any BasicCM service. The concrete type is service:BasicCM:icpname meaning that it is the specific type of BasicCM service.

3.2.5 Service Attributes

The Service Templates define the hallmarks associated to a given service and values that can be allotted. An example of an attribute is:
DTD=http://<ip>:<port>/ the attribute consists of an id and a value where the DTD is the id and the rest is the value.

4 How SLP Works

4.1 SLP API

One of the most important pieces in SLP specification is the Application Programmers Interface (API) information. It is an interface that allows programmers to use the protocol in their own application to locate services. An API helps developers to easy put SLP based features into their programs.

SLPReg
Register a service URL and its attribute with SLP.

SLPDeReg
Deregister an already registered service.

SLPFindSrvs
Finds services based on the services type or attribute.

SLPFindAttrs
Asks for a list of attributes for services that are registered with SLP.

SLPFindSrvTypes
Asks for a list of service types that has been registered with SLP.

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

4.2 Messages

To be able to create a framework and to show where the services are located, SLP agents communicate with each other trough eleven different messages. The dialog between the agents is often limited to very simple changes of queries and response messages.

Service Request (SrvRqst)
Messages from UAs to SAs and DAs with queries of a service location.

Service Reply (SrvRply)
Message from SAs and DAs that answers a SrvRqst. SrvRply consists of the requested service URL.

Service Registration (SrvReg)
Messages from SAs to DAs, with information about a service that is available.

Service Deregistration (SrvDeReg)
Messages from SAs to inform DAs that a service is no longer available.

Service Acknowledge (SrvAck)
A general acknowledgment from DAs to SAs as an answer on a SrvReg and SrvDeReg message.

Attribute Request (AttrRqst)
Message from UAs that requests the attributes of a service.

Attribute Reply (AttrRply)
Message from SAs and DAs to answer an AttrRqst. AttRply consists of a list with the attribute that where queried.

Service Type Request (SrvTypeRqst)
Messages from UAs to SAs and DAs with a request for service types that are available.

Service Type Reply (SrvTypeRply)
Message from SAs and DAs as an answer to a SrvTypeRqst. SrvTypeRply consists of a list with requested service types that are available.

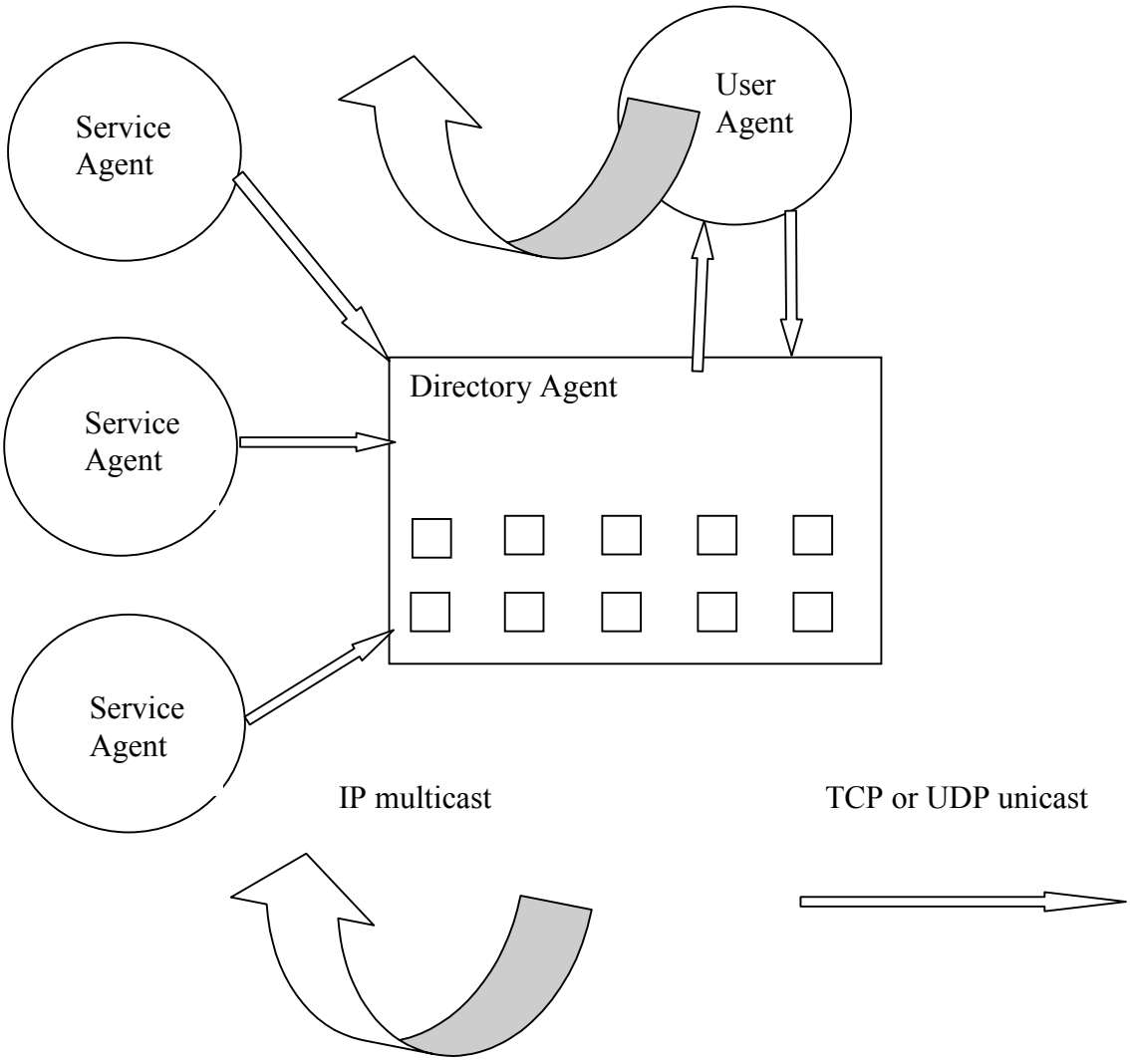
DA Advertisement (DAAdvert)
Message from DAs to let SAs and UAs know where they are located.

SA Advertisement (SAAdvert)
Messages from SAs to let UAs know where they are located.

Ericsson Erisoft 9/S94:002 Rev E

4.3 SLP Architecture

When the UA makes multicast requests for DAs it is only DAs configured with the same scopes that answers the request. If the DA is already known it does not answers the multicast request. If a DA is present the UA can use UDP or TCP to unicast requests to the DA. SAs can also register services to a known DA with unicast. The picture below shows the message flows.



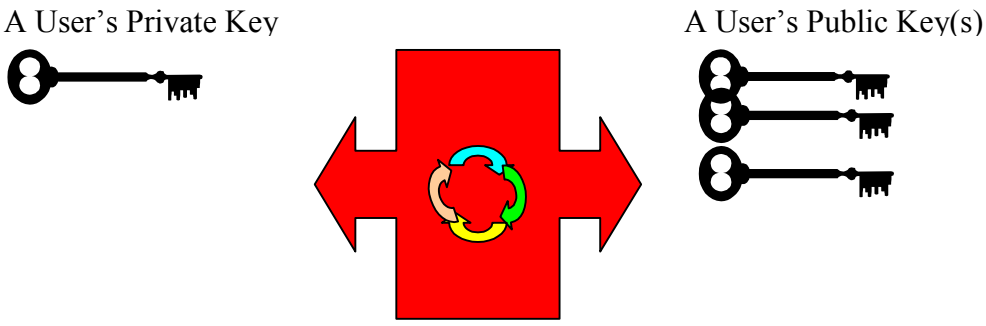
4.4 Security

SLP V2 has been designed to be a safe protocol. When the protocol is correctly implemented, it can assure integrity and reliability in transferring data between SLP agents.

An UA-agent or SA-agent does only open a connection to a remote agent such as a SA or DA when it is necessary, when information is required.

Objects authentication is the mechanism of the protocol security. Objects included in SLP messages and in some case the message it self is the protected material. It is important that the receiving agent can assure that the message is the same one that was sent by the transmitting agent. For example the DA needs to know that the service advertisement sent in a SrvReg is from a trusted SA and not an invented one that is trying to swindle an UA to use its phony service. An UA also needs to verify that a service advertisement was not a way for an opponent to steal the UAs information.

SLP uses a Private key and PKI (Public Key Infrastructure) to achieve that.



Public key technology is based on key-pairs. Users can use their private key to decrypt data encrypted using the corresponding public key.

There can be multiple copies of a user's public key, but only one copy of the private key, which is held by the user.

PKI assure following features:

- The person sending the transaction is actually the originator.
- The person receiving the transaction is the intended receiver.
- Data integrity has not been compromised.

Ericsson Erisoft 9/S94:002 Rev E

The SA is configured with a private key to calculate digital signatures for the object to sign. UAs and DAs are configured with SAs public key to verify the signature. The DA verifies the signature to make sure that it comes from a trusted SA and stores it later, the SA forwards it to the UA when the UA makes a request for it.

SPI is a text string that describes the security parameters such as key length, cryptoalgorithm parameters and keying material.

The goal of SLP is to provide service advertisement information to all users and applications that need it. Because of that, security can be difficult to achieve in all situations. Sometime preconfiguration of the agent's security parameters may not be possible, then a certain level of implied trust must exist between the various agents.

It is the SLP protocol who manages the security.

Ericsson Erisoft 9/S94:002 Rev E

5 NMS

The NMS (network management system) controls many aspects in a modern cellular network. To achieve this, it supervises many parts and events in these kinds of networks. A cellular network contains a large number of components, systems and subsystems that generates a large number of events during operation.

NMS has three different areas of operations.

5.1 Network Supervision

NMS have an advanced fault management, which achieves that all the alarms generated in the network can be taken care of. All events are logged which simplifies the follow-up. Analysis and statistical evaluation is rendered possible thanks to the performance management in this system.

It also has an Integrated Network Manager. The Integrated Network Manager makes it possible for handlers, of almost every entity in the network, to be started on remote and without any knowledge about the network equipment. Many problems can easy be solved without dispatching the field force.

5.2 Workflow Support

Alarms are generated when something is wrong in the network. NMS starts to support the Network Operator's own processes when an alarm has occurred and does not stop until the problem has been taken care of. This includes trouble ticketing and work order handling.

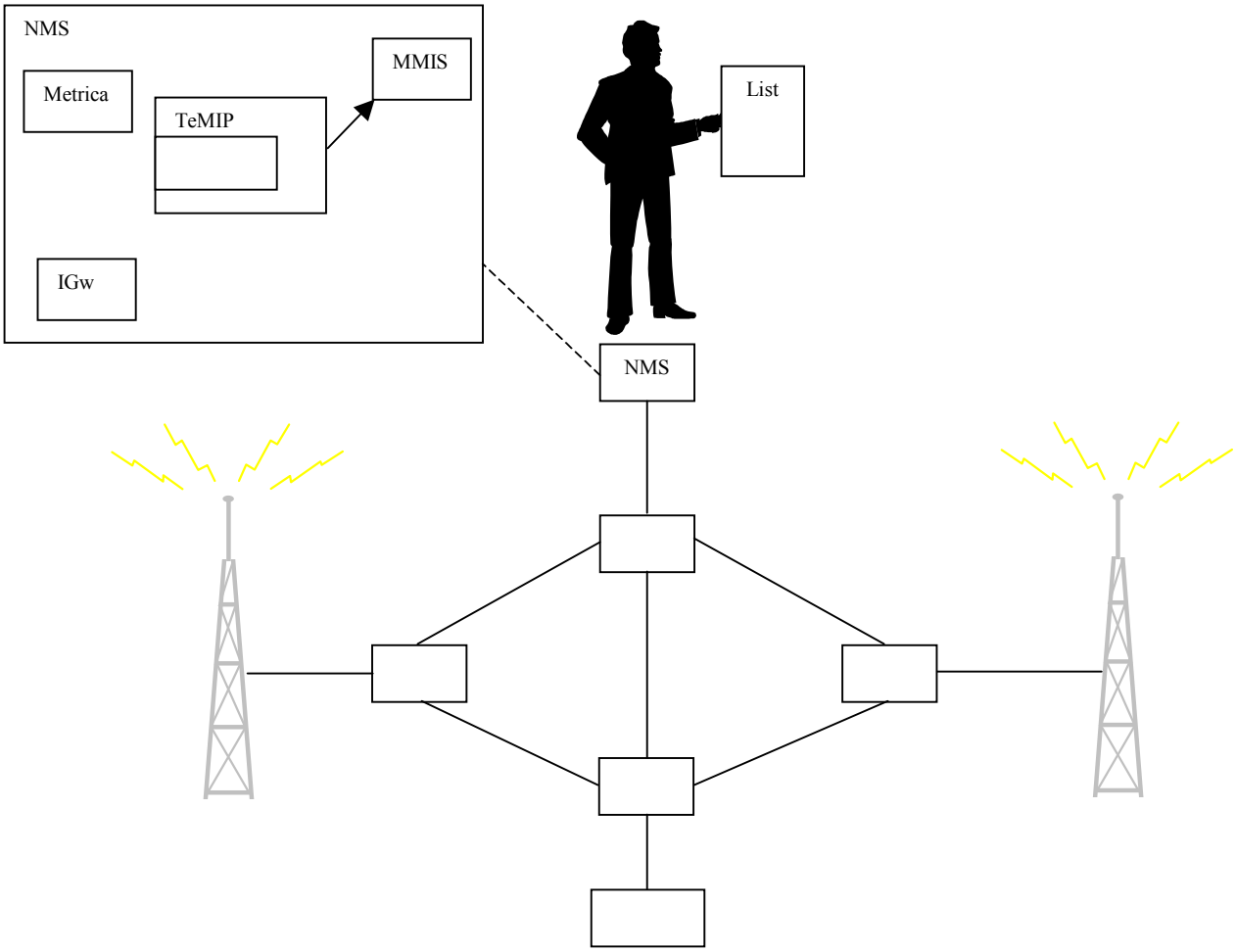
5.3 Equipment Handling

NMS keeps track of both component on-site and components in stock, even though there are thousands and thousands of individual components in a network. Each component's history can also be analyzed.

Ericsson Erisoft 9/S94:002 Rev E

5.4 An Example of NMS in a Mobile Network

This is an example of the edification of a cellular network that has included NMS.

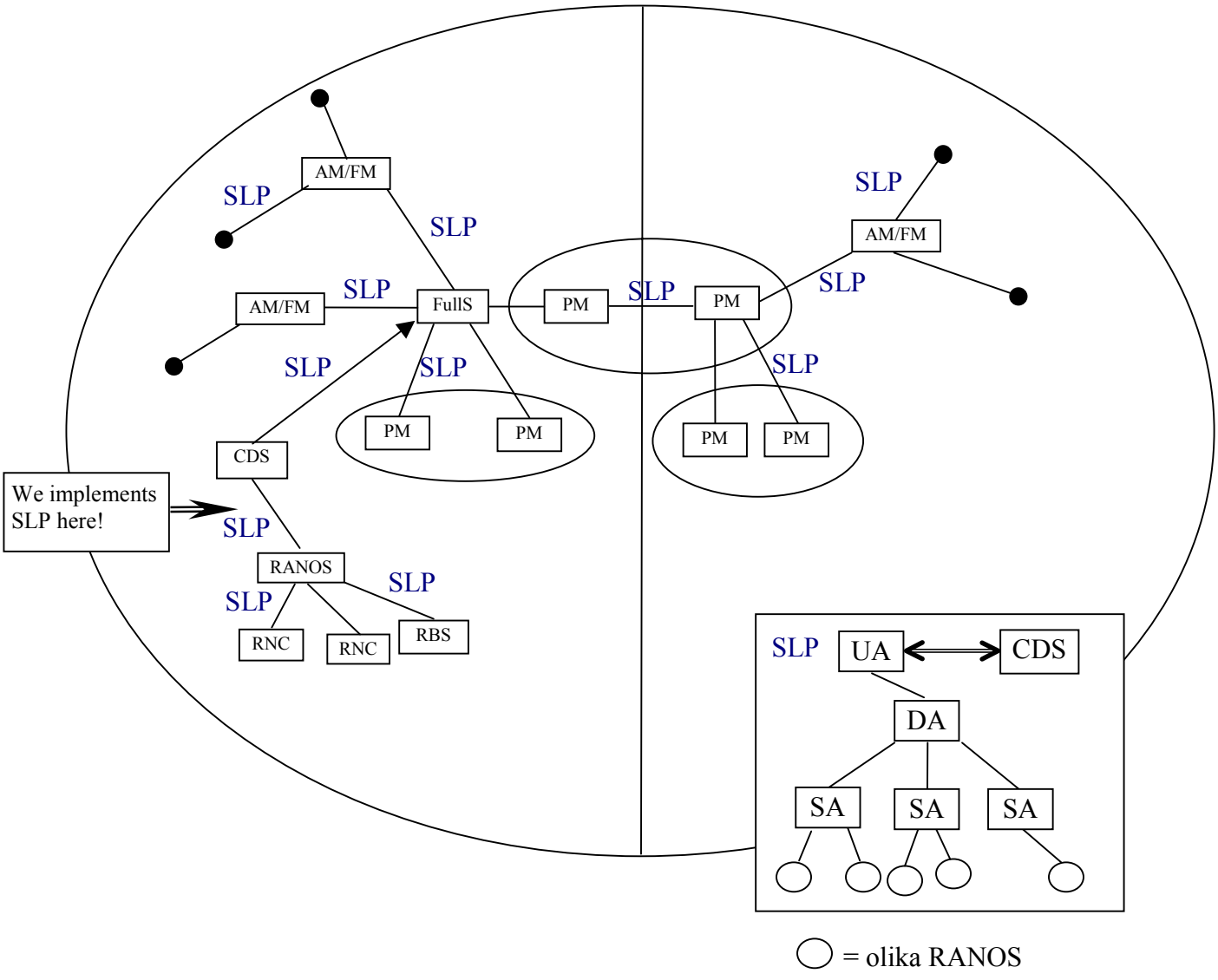


= Different Ericsson components in a mobile network, for example RNC, CN-OSS etc.

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

6 Integration of SLP into NMS

The picture below shows examples of where the Service Location Protocol can be implemented into a Network Management System to collect services.
The following explanations are collected from Ericsson:



6.1 Where do We Integrate SLP into NMS?

We had a discussion with our instructor and together we decided to integrate SLP between CDS and RANOS. This choice was made because we wanted to implement the protocol where Ericsson should have the most benefit and Ulrik thought that it should be at this place in the system.

6.1.1 CDS

Common Directory Server (CDS) is the part of a NMS system that makes requests for services. It gets the necessary information about the service from the user.

6.1.2 RANOS

Radio Access Network Operation Support (RANOS) is the part of a NMS system that keeps information about a service and wants to register that service.

7 Different Tools for Developing SLP

We have investigated the following tools for developing SLP.

- OpenSLP Version 0.9.0
- MiniSLP
- Solaris 8 slpd

The purpose of this was to find some tools that we could use as a gateway to reach the SLP API towards the SLP.

Special references and a short description of the documents to the tools can be found in Appendix D.

7.1 OpenSLP Version 0.9.0

Our research has been towards version 0.9.0 but the development has gone further and another release is already out on the Web.

The OpenSLP project is an effort to develop an open-source implementation of SLP suitable for commercial and non-commercial application.

It works on following operating systems:

- Linux
- BSD
- Solaris
- True64
- HPUX
- UnixWare OSR5
- Win32

Slpd is a daemon that provides SA functionality and with the right configuration DA functionality.

The SLP library provides UA functionality internally on a per process basis without the need to communicate with the daemon. Because of that, slpd must not always be loaded on every machine.

7.1.1 When is Slpd Needed?

- Slpd must be running on all machines that will be registering services. In other words, it is required on all machines that run applications that make calls to one of the following SLP APIs SLPReg(), SLPDeReg(), or SLPDelAttrs().

- It is also possible to register services from a file (/etc/slp.reg file), in that case slpd must also run if you expects the registration to be available on other machines.

- Slpd is required for automatic DA and scope discovery to work correctly. If you do not run the daemon, then DAs and scopes can only be discovered via DHCP or the (etc/slp.conf file) (Note: Due to a lack of a standard DHCP API DA discovery via DHCP is not yet supported).

7.1.2 When is Slpd not Needed?

- Slpd is not needed if a machine will only be requesting services. In other words, it is not required on machines if a call will never be made to SLPReg(), SLPDeReg, or SLPDelAttrs().

- The daemon is not needed on a machine if manual or DHCP DA or scope discovery is sufficient.

7.1.3 The slp.conf File

For every operating system except Win 32 the slp.conf file is located at /etc/slp.conf and contains configuration information that affects the operation of slpd and the applications that uses the OpenSlp libray. To discover where the file is located in Win32 read the Readme file.

7.1.4 The Static Registration File

It is often useful for users to be able to statically register services. Syntax for how the file should be written is specified in RFC 2614 that describes the syntax for a registration file. The default location for the registration file for all operating systems except Win32 is /etc/slp.reg and declared in the config file. The slpd reads the slp.reg file on startup and all of the registrations from the file are maintained by the daemon and will be registered as long as the slpd is alive. To discover where the file is located in Win32 read the Readme file.

7.1.5 Security

OpenSLP Version 0.9.0 supports the SLP V2 message authentication block to ensure that the registration is not modified during the transaction. The message authentication block also ensures that the messages are sent to and received by a valid agent.

7.2 Solaris 8 slpd

7.2.1 How SLP is Implemented

The SLP daemon:

- Employs passive and active directory agent discovery for all SAs and UAs on the local host
- A table of DAs is maintained and kept for the UAs and SAs on the local host.
- Reads the registration file and thus allows proxy registration.
- Can be configured to act as a DA or a SA.

The UA and SA libraries are for C/C++ libslp.so and for Java slp.jar and provide the UA or SA clients access to the SLP framework.

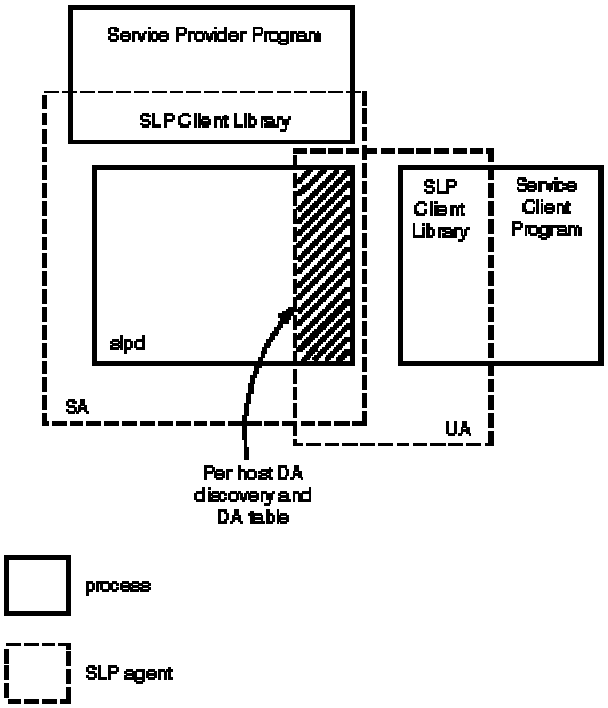
The client libraries main task:

- Provide the communication between a SA client and the slpd when registration and deregistration are required.
- Provide UA clients with the capability to make UA request.
- Provide communication of DA accessibility information between the daemon and UA clients.

Ericsson Erisoft 9/S94:002 Rev E

A figure showing how SLP is implemented in the Solaris environment.

The SA functionality is implemented by the SLP Client Library in the Service Provider Program. When to register or deregister a service the Service Provide Program uses the slpd.



The Library in the Service Client Program implements UA functionality.
The Program uses the SLP Client Library to issue multicast and unicast (to DAs) service requests and to query slpd for information on DAs. The daemon process takes care of all SA functionality, such as answering multicast requests, registering with DAs, and so on.

In Solaris, SLP is enabled by running the SLP daemon (slpd) located under /etc/init.d/slpd. Use start to start the daemon and stop to stop the same. Note that the daemon will only start if the configuration file exists. A SLP configuration example file is located at etc/inet/slp.conf.example. Rename this file to /etc/inet/slp.conf to enable the protocol at boot time.

7.2.2 Monitoring the SLP Traffic

Snoop is a passive administrative tool supplied by Solaris and useful for observing the traffic flow between hosts in a SLP enterprise. This is of use when determining whether any network or agent reconfiguration is needed.

7.3 Mini SLP

MiniSLP was developed in mars 1999.

The project can be built on the following platforms:

- Solaris 2.5 and higher using Sun's Visual Workshop or SPARCworks development environment
- NT 4.0 using Microsoft Visual Studio (5.0)
- Linux 2.0 and higher using the GNU development environment

MiniSlp supports SLP version2

The mslpd (MiniSLP daemon) works as a SA and mslplib (MiniSLP library) as a UA much like the other tools.

The release has not the following capabilities:

- Replies to SLPv2 Service Type or Attribute Request messages.
- Registration functions are provided using Serialized Registration files to the mslpd.
- Passive DADetection
- Supports of asynchronous requests.

No example program using the SLP API is available in this release.

There are two components in the Mini Service Location Protocol implementation: The mslpd and mslplib.

In MiniSLP the mslpd works as a SA and is a network daemon. Mslpd is loaded with a serialized registration file and optionally a configuration file.

Both the serialized registration file and the configuration file conform to the format specified in the API Internet Draft. The configuration file is further described in the source code.

mslpd.c	Describes the configuration properties which are understood by the mslpd.
mslplib_int.c	Describes the configuration properties which are used by the mslplib.
mslpd_reader.c	Describes the serialized registration files which are read in as service advertisements.

8 Prototype Developing

8.1 Prototype Requirements

To implement a prototype that shows the technology of SLP V2, the prototype should be developed with this protocol to solve the problem with "loose connections". Services should be able to be registered when they are created and they should not have to be predefined. This must be done because of the constantly changing characteristics and appearance of the service. It should also be done because of the constantly changing need of specific services. The user should not need to know where a service is located or what attributes it has. The only thing the user must know is what kind of service he/she requires and be able to choose one service of this type.

The prototype shall work between CDS and RANOS. The RANOS should be able to register or deregister a service. The CDS should be able to get a service that a RANOS has registered. It should also work between CDS and CN-OSS that is similar to RANOS but works in a different network.

8.2 Risks in the Project

In the beginning of our project we made an analysis and tried to define the major risks with the project.

8.2.1 Environment

A major problem could be if we lost our Solaris 8 environment. We did not want to lose the environment because SLP is a part of it and the risk of losing the environment where very big, because we had only borrowed it.

Another problem that we thought would be of great importance was the slpd written in Java. If the slpd written in Java was not finally developed we had to start implementing in C++ in the Solaris environment.

Ericsson Erisoft 9/S94:002 Rev E

8.2.2 SLP

The major problem of SLP we thought would be integration with already existing agents. We thought that it would be a problem because of the lack of understanding the implementation in the tools. We could only read the interfaces not the actual implementation. We really needed this to work, otherwise we would not have an application to test against the tools. In the worst case we would have been forced to implement our own UA, SA and DA.

Misunderstanding the protocol documentation could also be a problem. If we did not understand the intention we probably would have used it in the wrong way. Lack of experience in reading RFCs could have caused this problem. All the differences in the information about the SLP where another reason why this could have happened.

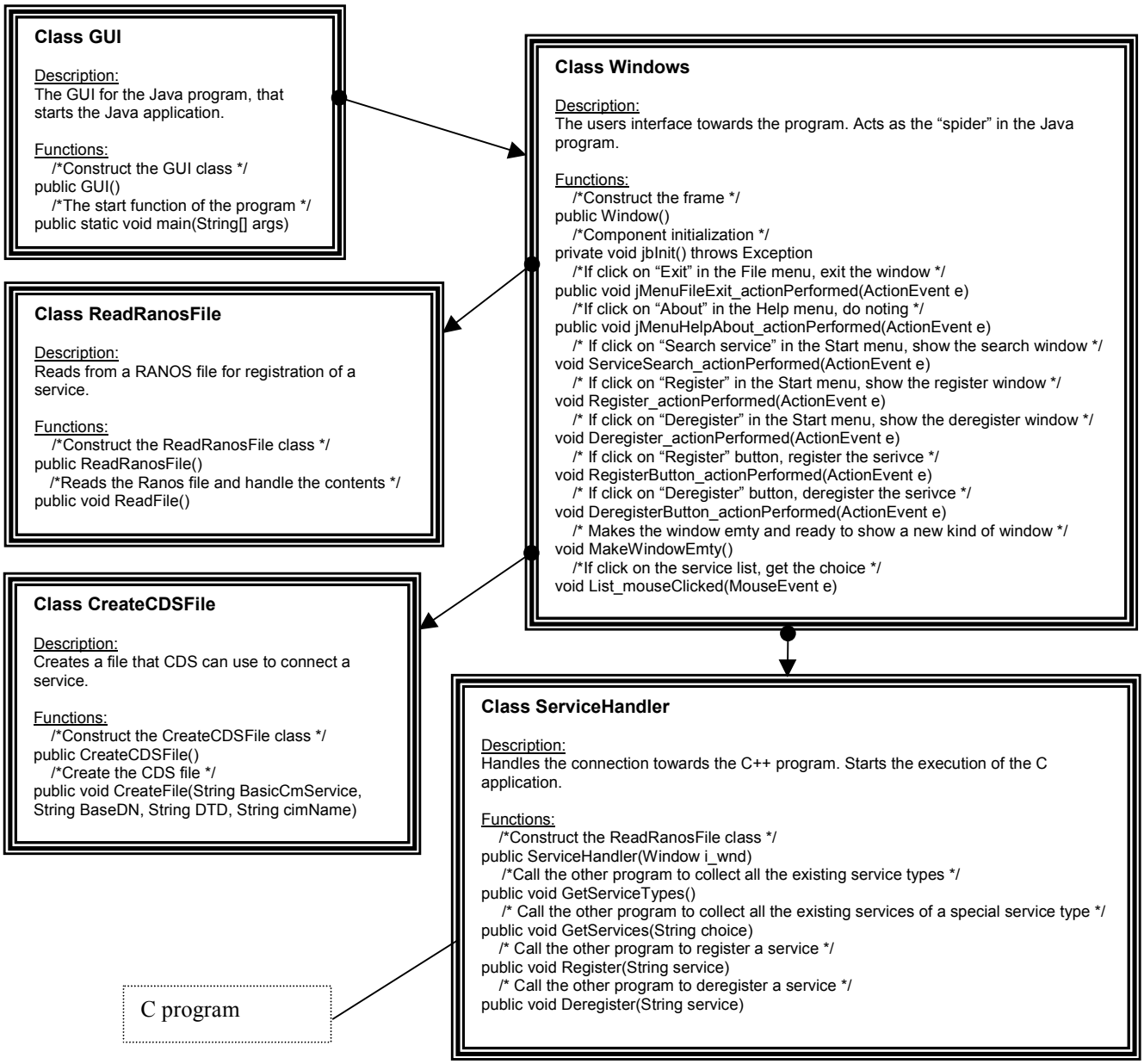
Ericsson Erisoft 9/S94:002 Rev E

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

8.3 Final Prototype Design

8.3.1 The Program Written in Java

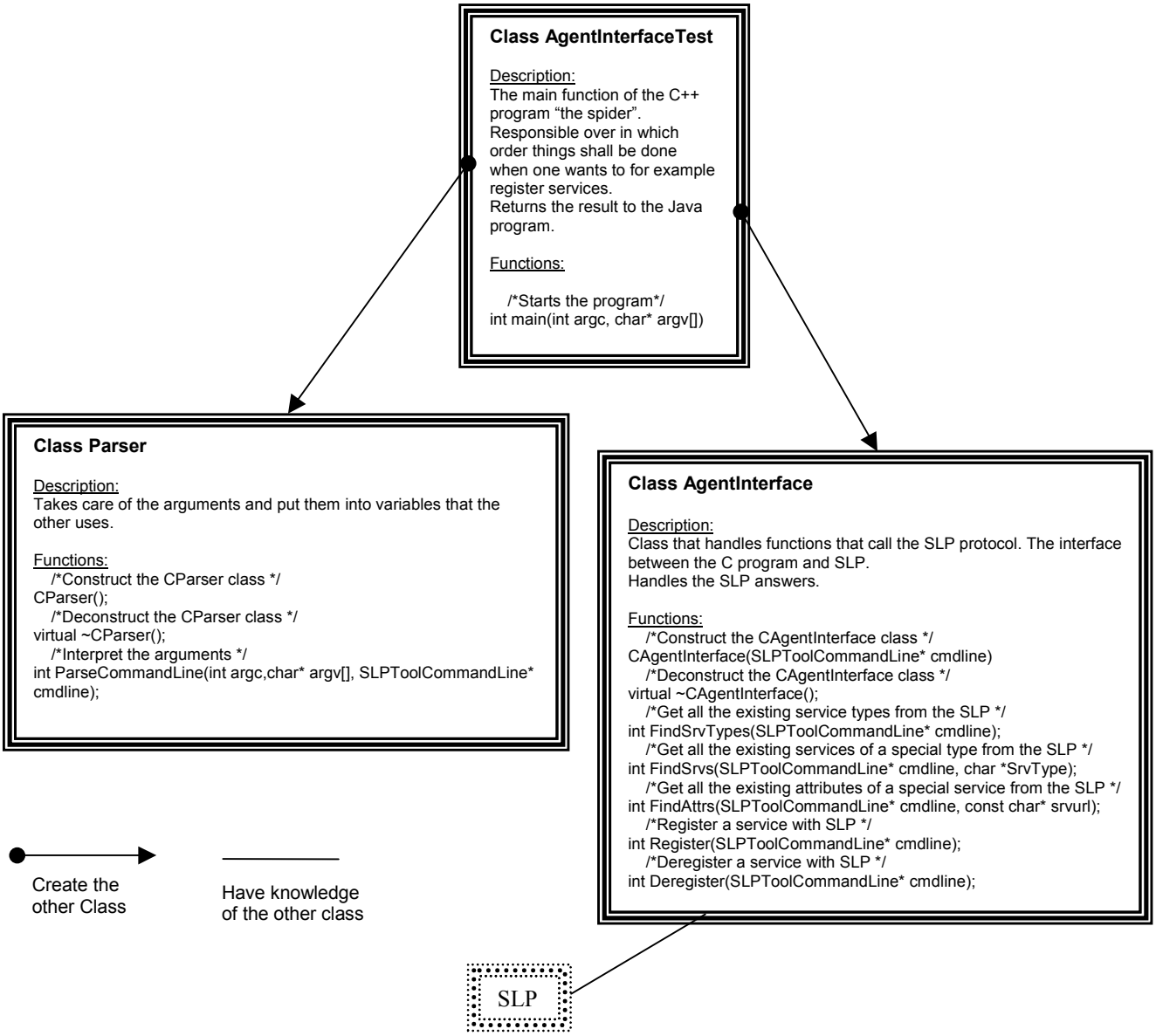
This program handles the connection between the User and the SLP, by forwarding instructions to the program written in C++. It is developed for all types of platforms and should be easy to use by the user.


Class GUI
Description:
The GUI for the Java program, that starts the Java application.
Functions:
/*Construct the GUI class */
public GUI()
/*The start function of the program */
public static void main(String[] args)
Class Windows
Description:
The users interface towards the program. Acts as the "spider" in the Java program.
Functions:
/*Construct the frame */
public Window()
/*Component initialization */
private void jbInit() throws Exception
/*If click on "Exit" in the File menu, exit the window */
public void jMenuFileExit_actionPerformed(ActionEvent e)
/*If click on "About" in the Help menu, do noting */
public void jMenuHelpAbout_actionPerformed(ActionEvent e)
/* If click on "Search service" in the Start menu, show the search window */
void ServiceSearch_actionPerformed(ActionEvent e)
/* If click on "Register" in the Start menu, show the register window */
void Register_actionPerformed(ActionEvent e)
/* If click on "Deregister" in the Start menu, show the deregister window */
void Deregister_actionPerformed(ActionEvent e)
/* If click on "Register" button, register the serivce */
void RegisterButton_actionPerformed(ActionEvent e)
/* If click on "Deregister" button, deregister the serivce */
void DeregisterButton_actionPerformed(ActionEvent e)
/* Makes the window emty and ready to show a new kind of window */
void MakeWindowEmty()
/*If click on the service list, get the choice */
void List_mouseClicked(MouseEvent e)
Class ReadRanosFile
Description:
Reads from a RANOS file for registration of a service.
Functions:
/*Construct the ReadRanosFile class */
public ReadRanosFile()
/*Reads the Ranos file and handle the contents */
public void ReadFile()
Class CreateCDSFile
Description:
Creates a file that CDS can use to connect a service.
Functions:
/*Construct the CreateCDSFile class */
public CreateCDSFile()
/*Create the CDS file */
public void CreateFile(String BasicCmService, String BaseDN, String DTD, String cimName)
Class ServiceHandler
Description:
Handles the connection towards the C++ program. Starts the execution of the C application.
Functions:
/*Construct the ReadRanosFile class */
public ServiceHandler(Window i_wnd)
/*Call the other program to collect all the existing service types */
public void GetServiceTypes()
/* Call the other program to collect all the existing services of a special service type */
public void GetServices(String choice)
/* Call the other program to register a service */
public void Register(String service)
/* Call the other program to deregister a service */
public void Deregister(String service)
C program

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

8.3.2 The Program Written in C++

This program handles all connections with SLP. It takes commands from the program written in Java, fulfills the mission and returns the result. This program has functionality for both an UA and a SA. The SA part handles the registration and deregistration commands to the protocol. The UA part handles the collection of service types and the services from the SLP.


Class AgentInterfaceTest
Description:
The main function of the C++ program "the spider".
Responsible over in which order things shall be done when one wants to for example register services.
Returns the result to the Java program.
Functions:
/*Starts the program*/
int main(int argc, char* argv[])
Class Parser
Description:
Takes care of the arguments and put them into variables that the other uses.
Functions:
/*Construct the CParser class */
CParser();
/*Deconstruct the CParser class */
virtual ~CParser();
/*Interpret the arguments */
int ParseCommandLine(int argc,char* argv[], SLPToolCommandLine* cmdline);
Class AgentInterface
Description:
Class that handles functions that call the SLP protocol. The interface between the C program and SLP.
Handles the SLP answers.
Functions:
/*Construct the CAgentInterface class */
CAgentInterface(SLPToolCommandLine* cmdline)
/*Deconstruct the CAgentInterface class */
virtual ~CAgentInterface();
/*Get all the existing service types from the SLP */
int FindSrvTypes(SLPToolCommandLine* cmdline);
/*Get all the existing services of a special type from the SLP */
int FindSrvs(SLPToolCommandLine* cmdline, char *SrvType);
/*Get all the existing attributes of a special service from the SLP */
int FindAttrs(SLPToolCommandLine* cmdline, const char* srvurl);
/*Register a service with SLP */
int Register(SLPToolCommandLine* cmdline);
/*Deregister a service with SLP */
int Deregister(SLPToolCommandLine* cmdline);
Create the other Class
Have knowledge of the other class
SLP

8.4 SLP Technology Test Programs

We started this part of our exam work with research after some implementation environments that seemed to fit our intentions. We also implemented a few test programs with help from some of the tools that we had found earlier in the research part. The purpose was to understand the construction of SLP, how it is supposed to work and be implemented. The result of this should be a prototype to test the technology with.

8.4.1 Implementation Environment Research

First we made a list of different environments that we had worked with earlier and that our supervisor thought could be usable.

We choose between the following environments:

- Forte J 4.0 (Java programming)
- JBuilder 4.0 (Java programming)
- Java Café (Java programming)
- Visual Studio C++ 6.0 (C/C++ programming)

We did not know what language that should best fit our assignment, so we decided to have both a C++ and a Java environment available for tests.

Visual Studio C++ 6.0 for C++ was a natural choice, because we been implementing in this environment before. It is a rather easy and a fairly good tool for developing programs in C or C++.

It was the first time that we tested Forte J 4.0 and we think that it is a rather complicated environment. It was difficult to understand and did not resemble of anything that we had tested before. We did not choose this one because of the difficulty in understanding it. Our conclusion was that it should have taken us too many hours before we would understand the environment good enough for our purpose.

Java Café never became a real option because Erisoft did not have any license to give us.

We chose JBuilder 4.0 because it was not that difficult to understand and it resembles the developing environments that we have already worked with.

Ericsson Erisoft 9/S94:002 Rev E

8.4.2 Developed Programs

We developed several programs before we could be satisfied with our technology-testing prototype.

8.4.2.1 SLP Implementing Test

The first test program we made was implemented in Java after a printer example from the book "Service Location Protocol for Enterprise Networks". "J. Kempf and P. StPierre 1999" The purpose of this program was to see how we should be able to use already existing SLP and what interfaces exists towards this protocol.

Unfortunately we never got this example to work because it was impossible to procure a SLP API in Java for windows. This API has not been developed yet.

We also wrote a program that could read a registration file. The purpose of this program was to understand how a SA reads a registration file.

8.4.2.2 Prototype in C++

The first real prototype was developed in C++ and towards OpenSLP-0.9.0 that is a SLP API in C for windows. We developed this program with a great deal of help from an SLP-Tool program that we found on the Internet. This tool was developed to help others to build their programs towards the OpenSLP.

The prototype worked in NT4. Observe that it demands that OpenSLP-0.9.0 is installed and started on the computer if register of services is required.

Ericsson Erisoft 9/S94:002 Rev E

8.4.2.3 Prototype in Java 2

Even though the first prototype worked, we decided to make a second prototype. This prototype should be developed in Java so that we could get a program that was independent on what operating system we used. The purpose of this was that we wanted an easy tool to develop a graphic user interface for Solaris 8. The main task was that the prototype should work in Solaris, because CDS and RANOS execute in this environment.

We rewrote the prototype in C++ to one in Java. The base in both these programs is the same, it is only the programming language that separates them. Solaris 8 has a SLP API for both Java and C and slpds for both languages, which means that we could use the same functions and SLP-calls as in the program written in C++ against OpenSLP.

This prototype worked very well on one computer but unfortunately not in a network. The problem is caused because the SLP tool written in Java is not yet complete.

8.4.2.4 The Final Prototype

The final prototype became a combination of a program written in C++ and a program written in Java. We chose to use the SLP tool written in C in Solaris 8 instead, because this tool is more developed. Our plan was to use SLP from a program written in C++ and still use Java to handle the graphics. Together these programs should become the final prototype that works in NT 4 with OpenSLP-0.9.0 and Solaris 8 with its tool written in C.

To achieve this we did the following things:

- We rewrote our first prototype, which already was written in C++, so that it could be started by the program written in Java.
- We developed software in Java, that handled all the graphics and became the interface between the user and the program written in C++. This software also started the program written in C++ and told it what to do.
- The programs are communicating with each other trough a specific file.

Ericsson Erisoft 9/S94:002 Rev E

This program does not work to 100 percent because it needs more development.
However it shows the purpose of SLP and is an idea how Ericsson Erisoft can use this protocol for there intentions.

8.5 Prototype Tests

On the basis of our test specification we tested our final application and discovered following:

8.5.1 Configuration

Configuration of a SA and a DA led to following result:
(For the configuration see Appendix A.)
Our application worked both as SA and DA in NT 4 environment, but in Solaris 8 we never got the chance to make a real test. We never succeeded to receive a DAAdvert() when we configured a DA in Solaris 8.

8.5.2 Registration

Registration of services led to following result:
(For a register file see Appendix B.)
Our application can register from a registration file in NT 4 environment, but in Solaris 8 we never succeeded with that. A real test has never been properly performed in Solaris 8. Observe this registration is only possible when the slpd is stopped and restarted. The registration file is only read by the slpd when it is started.

Registrations from a simulated RANOS file led to following result:
(For an RANOS file see Appendix C.)
Our application can register from a simulated RANOS file in NT 4, no test in Solaris has been done.

8.5.3 Services

Collection of service information led to following result:
If no services are available, a message occurs for the users.
It is possible to get service types and services of a specific type in both environments.
Our intention was that it would be possible to fetch services based on at least one specific attribute. This has never been possible to achieve because that feature has not yet been implemented in any of the tools that we have tested.

8.5.4 Errors

Error messages will occur in following situations:
When the program did not succeed in register or collect services error messages would occur. There is no possibility, as far as we can see, that the user can do anything that would make our program crash, but it is best not promising too much. We have tried to achieve that by not giving the user too many opportunities.

8.5.5 Networks

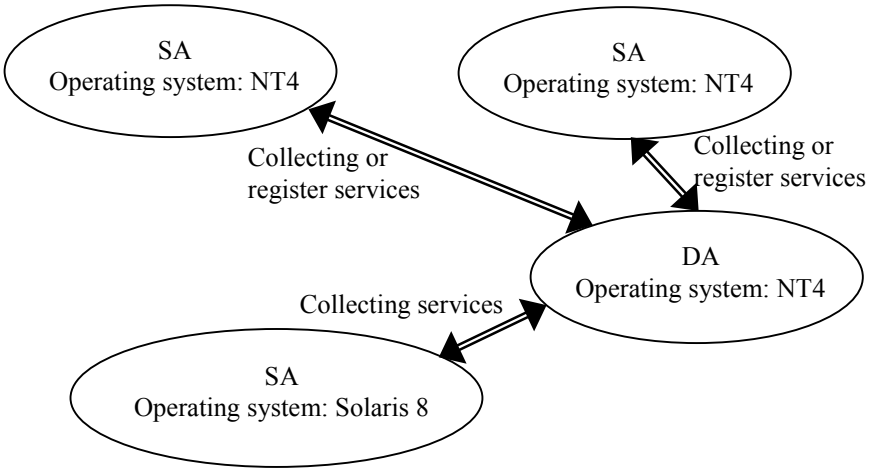
We have executed our program in both the personal net and the laboratory net.
We have also succeeded in communicating between these networks with our application.
In the laboratory net it has been difficult to achieve a working environment including our application.

Ericsson Erisoft 9/S94:002 Rev E

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

8.6 The Final System Design

Our prototype is executed on one Solaris 8 machine and on at least one NT4 machine. One NT4 machine is configured as an DA and the others as SAs. The NT4 machines can both register and collect services. The Solaris machine can only get services.



9 Conclusions

9.1 Tools

There are some tools available for developer of SLP. Among them we have chosen three that we have examined mSlp, OpenSLP and the SLP tool included in Solaris 8.

9.1.1 Functionality

We believe that the different tools works in the same way. They all have a slpd that acts as a SA or DA. All of the tools also have a library with interfaces. How far the different tools where developed differs. OpenSLP has already a new release and this shows that the development goes on. We do not have any clue about the others.

Documentation, availability and how easy it was to understand.

9.1.2 MSLP

The documentation was spartan and not so easy to find. We wished that the documentation would have been more complete and perhaps a user manual. The program was easy to find and install but how it worked was another question. We did not really understand what the program was doing or how we should use it. Some of the problems with the program where how to start and stop it, how to configure etc. The program consists of many test programs whose purpose where to show the functionality. Unfortunately we did not understand them either.

9.1.3 OpenSLP

The documentation was easy to find and the explanations satisfied us. A user guide, a developers guide could be found and those where of great help for us during the implementation and configuration. No problems to find and install the program, but there where some difficulties in starting and stopping the slpd. After some assistance from our instructor the problem disappeared. It was rather easy to start using openSLP. A tool where included in openSLP that showed the functionality and how we could use openSLP. The tool was easy to understand.

Ericsson Erisoft 9/S94:002 Rev E

9.1.4 SLP in Solaris 8

To our surprise Solaris 8 has SLP implemented by default. That solved a lot of problems for us because we had decided that our prototype should work in Solaris. The documentation is complete but sometime it has been difficult to directly find what we were looking for. The documentation was easy to find because Solaris 8 has a SLP manual. Non of the APIs where especially difficult. The SLP tool written in Java caused us a lot of problems because it was not developed enough. We also have had great problems in knowing if the slpd has been started or stopped.
We have unfortunately sometimes succeeded to kill the process not knowing why it happened.

9.1.5 Reasons for Chosen Different Tools

The reasons why we decided not to use mSLP are as followed.
Difficulties in understanding mSLP are the main cause why we did not choose it.
Another aspect was that mSLP did not have all the functionality that we wanted.
To be honest we never gave it a real chance. The version that we found was from 1999 and we did not find any continue so we thought that it was too old. Perhaps this is a suitable tool for our purpose if the circumstances had been different.

As mention earlier the OpenSLP documentation had mentioned enough details that we felt confident enough to try it. OpenSLP worked almost as we had predicted except from the fact that OpenSLP did not support collecting services on the basis of attributes. On the other hand our instructor did not think that this was an important function. Another reason why we choose to use OpenSLP was the easiness to start using it. The already included test tool gave us such guidance so it became very easy to implement a prototype.

It was a natural choice that we decided to use SLP in Solaris 8 because this where the main environment. We tried to use OpenSLP in Solaris 8 but we did not manage to get it to work properly so the ordinary slpd became our only option.

Ericsson Erisoft 9/S94:002 Rev E

9.2 Our Application

Because of lack of time we never fulfilled our assignment towards Ericsson. We have not managed to test our application towards a simulated RANOS and CDS.

This is what we manage to accomplish.

- Register a service works in NT 4 but not in Solaris 8.
- Deregister is not complete implemented.
- Get services works in both environments but our version demands four attributes.

We managed to use one Solaris machine as a DA at one occasion only after that we have just been able to have NT 4 machines as DAs. The reason for that is something that we can not explain only speculate about. A possible cause can be the instability of the operating system on our Solaris machine. Another reason is may be that we made something wrong during the configuration of DA and SAs or that the slpd environments differ from each other. The registration works as mention earlier in NT 4 and can then be found in the other operating system, it does not work the other way. We think that this can derive from our program or once again that something differs between the different slpds. Because we have not handled the registration problem in Solaris 8, the program does not work according the requirement specification.

Another aspect to consider is that we have implemented and executed our program in a sterile surrounding. We do not know how stable the prototype, protocol or daemons really is when the system is being normally used. This thought occurred to us after a mistake had been done, someone else was executing applications at the same time on the same computer as we where running our prototype.

9.3 Multicast and Firewalls

The firewalls in Ericsson networks probably caused that multicast did not work in our tests. As a result of that we had to specify a DA with a specific IP address and configure the SA to call the DA on that address. This is to static and not the best solution, all depending on what the company wants.

9.4 Predicted Risks

The loss of Solaris 8 environment unfortunately came true, because the machine was needed by Ericsson employees. Therefor we can not present a real test of our program between different machines consisting only of this environment. Teamwork between Solaris 8 and NT4 became the solution.

Our first intention was to build our program in Java because that would make it easier to connect against already existing components, but the SLP tool written in Java where not sufficient developed, therefore as a solution of that we made a combined prototype consisting of both C++ and Java. The reason why we chose to use this instead of only C++ where to keep the two languages advantages.

We thought that the integration with already existing UA, SA and DA would be a problem. Thanks to a test tool that we found together with openSLP this has not as far as we can see been a problem.

In the beginning it was some problems in understanding the RFCs but we think that we have translated them correctly

9.5 Final Conclusions

One of our assignments was to find some tools that would be of benefit for Ericsson when using the Service Location Protocol.

In NT 4 we have not noticed any problem with OpenSLP but security is not something that we have tested and therefore can not guarantee. Our experience tells us that an object-orientated language is better than plain C so the future developer with OpenSLP may consider that it is written in C. Despite of that we believe that this can be a usable tool in this environment. It keeps functionality to in a dynamic way register, deregister and collect services and this should achieve a solution of loose connections, for example changing or upgrading of components without interference in the integration. The biggest benefit is the easiness to use this tool.

The SLP in Solaris 8 has given us problems, but we can not explain what caused them. Many components in this environment, for example one machine has always been unstable and the other machine was taken away from us after just one test, contribute to the reason why we can not make a final conclusion about this tool.

The idea with Service Location Protocol is very good. The dynamically discovering the existence, location and configuration of services is something that we believe would benefit Ericsson. The task to make the integration of components easier is

Ericsson Erisoft 9/S94:002 Rev E

something that SLP never can achieve, but configuration of the components will become easier.

SLP V2 is supposed to be a safe protocol, due to the fact that the receiving agent can assure that incoming messages are not corrupted or changed. The receiving agent is also capable of determining who the sender really is via the authentication.

Because SLP will use multicast as default the solution for work between different network with a firewall will be:

- Statically determine where the DA is located and make all communication through unicast.
- Only share services within a network and no access to services in other networks.
- Not have any firewalls at all between the different networks, something that is not recommended.
- Configure the firewalls to pass through some traffic, for instance multicast.

Despite of all benefits, that we believe SLP would give, we are forced not to recommend it because it is only a purposed standard and not recommended into a disruption-sensitive environment. When it becomes a standard we believe that the Service Location Protocol is something that we can recommend.

9.5.1 SLP in the Future

As mention earlier SLP is still under development. The present goal is to advance SLPv2 to the status of "Draft Standard". A Draft Standard is normally considered to be a final specification. It must be well-understood and known to be quite stable.

Present project in SLP developing are:

- mSLP Mesh enhanced SLP v2 in Java
- OpenSLP a version 1.0.0 has already been released and continuous developing.
- LDAP SLP: SLP implementation supporting SLP v2 with an LDAP back end.

There are products that in this moment already are using SLP for instance.

- Printers such as HPand Minolta support SLP in their network enabled printers.
- Platform support for SLP such as Sun Microsystems, Novell

The fact that we have concentrated us to implement SLP between CDS and RANOS does not mean that SLP can not be implemented elsewhere in a Mobile Network. We believe that SLP can be implemented everywhere where services and configurations in networks are needed. For instance network printers, fax-machines and so on.

Ericsson Erisoft 9/S94:002 Rev E

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

10 Glossary

Abstract Service Type: A name used to designate services types with multiple different protocols. A hypothetical example is "service:printer", with concrete types "service:printer:lpr" or "service:printer:http".

AM: Access Module. A module in the TeMIP system. The AM is a link between the TeMIP system and the 'outside world'.

API: Application Programmers Interface

Attributes: A collection of tags and values describing the characteristics of a service.

Broadcast: A data link layer network transmission procedure required transmitting a packet to every machine on a subnet. Broadcast packets are typically not routed beyond the subnet.

CDS: Keeps the topologic data from many different RANOS, knows which net-elements that are present. It also consists of how the different parts are connected in a whole mobile network.

Concrete Service Type: A service type name containing an abstract types name and its protocol or generic URL name. A hypothetical example is service:printer:lpr

Directory Agent(DA): A process that caches SLP services adverticement registred by service agents and forvards the service adverticement to user agents on demand.

DHCP: Dynamic Host Configuration Protocol. A broadcast protocol that allows a machine to discover its IP address based on its MAC address and certain other information.

FM: Function Module. A kind of TeMIP module adds functionality to the system.

FullS: Full Servicer (Includes AM, FM and PM). A nod that are capable to run AM/FM/PM

Generic URL: A URL having a scheme different from the service:scheme.

Ericsson Erisoft 9/S94:002 Rev E

GPL:	Gnu General Public License
Hostname:	The internet name of a specific computer.
IEFT:	Internet Engineering Task Force. A volunteer group of engineers and scientists who are involved in developing proposals for protocols and procedures to enhance the Internet.
IP:	Internet Protocol. The network layer protocol used in the global Internet and also in many corporate and private internal networks.
Linux:	A UNIX based operating system, available source-code distributed under GPL.
MSC:	Mobile Switching Center. The switch between the fixed and the mobile network.
MSLPd:	Mini Service Location Protocol daemon.
Multicast:	A network layer (IP) procedure by which a datagram packet can be sent to multiple machines within an IP network, without requiring every machine on the network to handle the packet, as is the case for broadcast. Multicast requires that routers can be configured with special routing protocols.
NMS:	Network Management Solution.
NT4:	An operating system version developed by Microsoft.
OpenSLP:	The open source Service Location Protocol project.
PM:	Presentation Module. A module in the TeMIP system. The PM is a link between the TeMIP system and the user.
PKI:	Public Key Infrastructure. The supporting software on a programming platform that allows easy administration and use of public key cryptography. Also a framework for providing applications with public keys

Ericsson Erisoft 9/S94:002 Rev E

RANOS: A supervising system that handles the mobile net and a bridge used for collecting information from 3G about available services.

RBS: Radio Base Stations. A name used for mobile telephony base stations.

RFC: Request for Comments. A proposed standard for changing or enhancing IP networks.

SA server: A process on a host machine running SLP that takes care of all off-machine communication for any client Sas. The SA server allows multiple clients SA processes to run.

Scopes: In SLP, an administrative or logical grouping of services. Ex If you are in Sweden the default value for scopes are sw, (language) SWE (Country).

Service Access Point: The network address and port number of a process offering a service.

Service advertisement: Information describing a service: A service advertisement consist of the service type, service access point, lifetime, and attributes. A service advertisement is a URL and a collection of attribute/value list pairs that describe a service. All service advertisements have a lifetime. After the lifetime expires, the service advertisement is no longer valid, unless reregistered.

Se*rvice Agent (SA):* An SLP agent acting on the behalf of a service. The service agent distributes service advertisements for the service on request and registers the service with any directory agents.

Service Location Protocol (SLP): A proposed standard protocol for locating services on IP networks.

Service:scheme: The URL scheme for service URLs having the service:scheme are advertised by SLP, but generic URL can be as well.

Service Type: The name of a network service. An example of a service type name is *"service:printing:lpr"*

Service Type Template: A formalised, computer-readable description of a service type.

Ericsson Erisoft 9/S94:002 Rev E

Slpd:	Slpd is a daemon process that acts as either a DA or an SA server. Service processes on a host register their service advertisements with slpd rather than maintaining the advertisements individually. These service processes contain a SA client library that communicates with slpd, acting as the SA server. slpd forwards all registrations and deregistrations to DAs, and times out expired service advertisements. slpd also maintains a table of available DAs by performing active and passive DA discovery. Through these mechanisms, DA information is provided to UA clients. UA clients use slpd on their host only for DA information. Each SLP host automatically has an slpd daemon. You can optionally configure slpd as a DA.
SLP Port:	Port 427. Responds to both TCP and UDP.
SLP v2:	Version 2 of the SLP protocol, derived from SLP v1.
SMS:	Short Message Services. A part of the GSM that makes it possible to send text messages to a GSM mobile telephone.
Solaris:	Operating system developed by Sun.
SPARC:	A 32-bit Scalable Processor developed by Sun.
SPI:	Security Parameter Index. A token indicating the keying material, algorithm parameters, and other information needed to encrypt or decrypt a signature from an SLP authentication block.
SRVLOC WG:	Service Location Working Group. The subgroup of the IEFT that designed SLP
Sun:	A company that develops software.
Transmission Control Protocol (TCP):	Reliable transport layer protocol widely used in IP networks.
Unicode:	A two-byte character encoding widely used in modern operating systems. Unicode characters are not ideal for network protocols because they suffer from byte ordering problems.

Ericsson Erisoft 9/S94:002 Rev E

Unicast: Sends messages to one specific host.

URL: Uniform Resource Locator. A standard for writing a text reference to an arbitrary piece of data in the World Wide Web

URL Path: In a service: URL, that part of the URL after the service access point. The URL path contains additional information needed to contact the service.
A service URL is used to advertise the network location of services. It contains the service type name, host name, or network address of the system hosting the service, optional port number, and other information necessary to use the service.

User Agent (UA): An SLP agent acting on the behalf of a user or application. A user agent obtains service advertisement information for its clients.

User Datagram Protocol (UDP): An unreliable transport layer protocol widely used in IP networks.

Ericsson Erisoft 9/S94:002 Rev E

11 References

[1] The description over our "Task description" EPL-01: 000187 USV

[2] James Kempf and Pete St.Pierre, "Service Location Protocol for Enterprise Network"
ISBN: 0-471-31587-7 1999

[3] SLP WHITE PAPER TOPIC April 2001
http://playground.sun.com/srvloc/slp_white_paper.html

[4] Description of Working Group April 2001
http://www.ietf.org/html.characters/svrloc-charter.html

[5] Introduction to the Service Location Protocol April 2001
http://www.svrloc.org/intro.html

[6] An Introduction to SLP April 2001
http://www.openslp.org/doc/html/IntoductionToSLP/index.html

[7] Ericsson documents

[8] Service Location Protocol Administration Guide April 2001
http://docs.sun.com:80/ab2/coll.47.11/SVCLOCADMIN/

[9] OpenSLP Users Guide April 2001
http://www.openslp.org/doc/html/UsersGuide/index.html

[10] OpenSLP Programmers Guide April 2001
http://www.openslp.org/doc/html/ProgrammersGuide/index.html

[11] Mini Service Location Protocol Documents April 2001
http://www.cs.cmu.edu/~pach/mslp/documents.html

[13] Public Key Infrastructure (PKI) April 2001
http://www.cio.usmc.mil/c4i/cio/c4ibrief/sld001.htm

[14] J.Kempf, E.Guttman Sun Microsystems - June 1999
An API for Service Location RFC 2614

[15] J.Kempf, E.Guttman, C. Perkins Sun Microsystems - June 1999
Service Templates and Service: Schemes RFC 2609

[16] E.Guttman, C. Perkins Sun Microsystems, J.Veizades @Home Network, M. Day Vinca Corporation - June 1999
Service Location Protocol, Version 2 RFC 2608

[17] S. Bradner Harvard University - June 1996
The Internet Standard Process – Revision 3 RFC 2026

Appendix A – Configuration of DA and SA

A.1 SA Configuration in Solaris 8

```
#################################################################################
#
# OpenSLP configuration file
#
# Format and contents conform to specification in IETF RFC 2614 so the
# comments use the language of the RFC.  In OpenSLP, SLPD operates as an SA
# and a DA.  The SLP UA functionality is encapsulated by SLPLIB.
#
# Observe that the proxy registration file setting is not available.
# The file is always read when the slpd is started. If no default registration is wanted
# mark all the lines in the registration file.
#################################################################################

#--------------------------------------------------------------------------
# Static Scope and DA Configuration
#--------------------------------------------------------------------------

# This option is a comma delimited list of strings indicating the only scopes
# a UA or SA is allowed when making requests or registering or the scopes a
# DA must support. (default value is "DEFAULT")
# net.slp.useScopes = default

# Allows administrator to force UA and SA agents to use specific DAs.  If
# this setting is not used dynamic DA discovery will be used to determine
# which DAs to use.  (Default is to use dynamic DA discovery)

 net.slp.DAAddresses = 150.132.181.22

#--------------------------------------------------------------------------
# Network Configuration Properties
#--------------------------------------------------------------------------

# Force broadcasts to be used instead of multicast. This setting is seldom
# necessary since OpenSLP will automatically use broadcast if multicast
# is unavailable. (Default is false)
#;net.slp.isBroadcastOnly = true
```

```
# A boolean indicating whether passive DA detection should be used.
# (Default is true)

 net.slp.passiveDADetection = false

# A boolean indicating whether active DA detection should be used. This is
# useful when the DAs available are explicitly restricted to those obtained
# from DHCP or the net.slp.DAAddresses property (Default is true)
# net.slp.activeDADetection = false

# A 16 bit positive integer giving the number of seconds between DA active
# discovery queries.  Default is 900 seconds (15 minutes).  If the
# property is set to zero, active discovery is turned off.  This is useful
# when the DAs available are explicitly restricted to those obtained from
# DHCP or the net.slp.DAAddresses property.

 net.slp.DAActiveDiscoveryInterval = 0

# A positive integer that is less than or equal to 255. (The default is  255)
#;net.slp.multicastTTL = 255

# An integer giving the maximum amount of time (in milliseconds) to perform
# active DA discovery requests. (Default is 2000 ms or 2 secs).
#;net.slp.DADiscoveryMaximumWait = 2000

# A value-list of 32 bit integers used as timeouts, in milliseconds, to
# implement the multicast convergence algorithm during active DA discovery.
# Each value specifies the time to wait before sending the next request, or
# until nothing new has been learned from two successive requests.
# Default is:  500,750,1000,1500,2000,3000.
#;net.slp.DADiscoveryTimeouts = 500,750,1000,1500,2000,3000

# An integer giving the maximum amount of time (in milliseconds) to perform
# multicast requests. (Default is 5000 ms or 5 secs).
#;net.slp.multicastMaximumWait = 5000

# A value-list of 32 bit integers used as timeouts, in milliseconds, to
# implement the multicast convergence algorithm.  Each value specifies
# the time to wait before sending the next request, or until nothing new
# has been learned from two successive requests.
# Default is:  500,750,1000,1500,2000,3000.  In a slow network the less
# aggressive values of 3000,3000,3000,3000,3000 allow better performance.
#;net.slp.multicastTimeouts  = 500,750,1000,1500,2000,3000
```

Ericsson Erisoft 9/S94:002 Rev E

```
# An integer giving the maximum amount of time (in milliseconds) to perform
# unicast requests. (Default is 5000 ms or 5 secs).
#;net.slp.unicastMaximumWait = 5000

# A value-list of 32 bit integers used as timeouts, in milliseconds, to
# implement unicast datagram transmission to DAs.  The nth value gives
# the time to block waiting for a reply on the nth try to contact the DA.
# Currently OpenSLP uses TCP for all unicast communication so this setting
# does not do anything
#;net.slp.unicastTimeouts  = 500,750,1000,1500,2000,3000

# To OpenSLP the following is the same as net.slp.unicastTimeouts.  Use
# net.slp.unicastTimeouts instead.
#;net.slp.datagramTimeouts = IGNORED

# An integer giving the maximum value for all random wait parameters.
# (Default is 5000 or 5 sec)
#;net.slp.randomWaitBound = 5000

# A integer giving the network packet MTU in bytes. (Default is 1400)
#;net.slp.MTU = 1400

# A list of IP address of network interfaces on which the DA/SA should listen
# for slp requests.  By default, slpd will use all interfaces.
#;net.slp.interfaces = 1.2.3.4,1.2.3.5,1.2.3.6

#--------------------------------------------------------------------------
# Security
#--------------------------------------------------------------------------

# A boolean indicating whether the agent should enable security for URLs,
# attribute lists, DAAdverts, and SAAdverts.  (Default is false and
# ENABLE_SECURITY code must be compiled)
#;net.slp.securityEnabled=true

# A boolean indicating whether the DA or SA will only allow deregistrations
# and re-registration from the *exact* host that made the registration.
# (Default setting if false and ENABLE_SECURITY code must be compiled).
#;net.slp.checkSourceAddr=true
```

```
#------------------------------------------------------------------------
# DA Specific Configuration
#------------------------------------------------------------------------

# Enables slpd to function as a DA. Only a very few DAs should exist.  It
# is suggested that the administrator read the OpenSLP users guide before
# enabling this setting (default is false).

 net.slp.isDA = true


#------------------------------------------------------------------------
# SA Specific Configuration
#------------------------------------------------------------------------
# Not yet supported by OpenSLP

#------------------------------------------------------------------------
# Tracing and Logging
#------------------------------------------------------------------------

# A boolean controlling printing of messages about traffic with DAs.
# Default is false.
#;net.slp.traceDATraffic = true

# A boolean controlling printing of details on SLP messages. The fields in
# all incoming messages and outgoing replies are printed.  Default is false.
#;net.slp.traceMsg = true

# A boolean controlling printing details when a SLP message is dropped for
# any reason.  Default is false.
#;net.slp.traceDrop = true

# A boolean controlling dumps of all registered services upon registration
# and deregistration.  If true, the contents of the DA or SA server are
# dumped after a registration or deregistration occurs.  Default is false.
#;net.slp.traceReg = true
```

Ericsson Erisoft 9/S94:002 Rev E

```
#------------------------------------------------------------------------
# Serialized Proxy Registration
#------------------------------------------------------------------------

# The net.slp.serializedRegURL is not supported by net.slp.serializeRegURL.
#
# slpd accepts the [-r] command line parameter that specifies the serialized
# registration file.  The default serialized registration file is
# /etc/slp.reg
```

A.2 SA Configuration in Solaris 8

```
#################################################################################
#
# ident     "@(#)slp.conf.example          1.3       99/10/19 SMI"
#
#Configure file for a SA in Solaris 8
# The file /etc/inet/slp.conf must exist in order for slpd(1m) to start,
# See slp.conf(4), slpd(1m), or RFC 2614 for more information.
# No properties are acutally set, so uncomment and edit any properties
# you wish to change.
#
#################################################################################
##
#--------------------------------------------------------------------------
# DA configuration
#--------------------------------------------------------------------------
#
# net.slp.isDA=true
#

#--------------------------------------------------------------------------
# Static scope configuration
#--------------------------------------------------------------------------
# Determines the set of scopes which this SA or DA supports. These
# scopes will also be included in the list returned from a find scopes
# query. This example simply sets the property to the default
# net.slp.useScopes=default
#
# This example sets the property to a site-specific list:
# net.slp.useScopes=default
#

#--------------------------------------------------------------------------
# Static DA Configuration
#--------------------------------------------------------------------------
# Forces UAs and SAs to use a set of DAs.
#
 net.slp.DAAddresses=150.132.181.22
#
```

```
#------------------------------------------------------------------------
# Serialized Proxy Registrations
#------------------------------------------------------------------------
# Specifies a file containing a set of registrations to be processed
# when slpd starts up, written as an URL.
# To load a proxy-registration file at /etc/inet/slpd.reg:
# net.slp.serializedRegURL=file:/etc/inet/slpd.reg
#

#------------------------------------------------------------------------
# Tracing and Logging
#------------------------------------------------------------------------
# By default, all tracing and logging is disabled.
#
# Display messages about traffic with DAs:
# net.slp.traceDATraffic=true
#
# Display details about SLP messages:
# net.slp.traceMsg=true
#
# Display details whenever a message is dropped:
# net.slp.traceDrop=true
#
# Dump service table upon registration or deregistration:
# net.slp.traceReg=true
#

#
# Set this property to change the default DA Heartbeat (10800 seconds).
# This example sets it to 7200 seconds (two hours):
# net.slp.DAHeartBeat=72
#

#
# Use this property to configure special attributes for DAs; these
# will be included in the attrs field in a DAAdvert. Currently
# the only useful property is min-refresh-interval, which allows
# you to set the minimum interval for which a DA will accept
# registration refreshes. This example sets the minimum interval
# to 3 hours (10800 seconds):
# net.slp.DAAttributes=(min-refresh-interval=10800)
#
```

```
#
# Use this property to configure special attributes for SAs; these
# will be included in the attrs field in a SAAdvert. This example
# sets the attribute "special-features" to "none":
# net.slp.SAAttributes=(special-features=none)
#

#
# Force SLP agents to use broadcast only (instead of the default multicast):
# net.slp.isBroadcastOnly=true
#

#
# Disable passive DA detection. Default is enabled.
 net.slp.passiveDADetection=false
#

#
# Change the multicast time-to-live (TTL). Default is 255. This example
# limits SLP multicast to the local subnet only:
# net.slp.multicastTTL=1
#

#
# Set this property to change the default interval between DA discoveries.
# Default is 900 seconds (15 minutes). This example changes it to
# 2 minutes (at the risk of increasing network congestion):
#
 net.slp.DAActiveDiscoveryInterval=0
#

#
# Set this property to change the maximum amount of time to wait until
# multicast convergance completes. Default is 15000 ms (15 seconds).
#This example changes it to 20000 ms (20 seconds), which may be
#desirable on congested networks with many SAs:
#
# net.slp.multicastMaximumWait=20000
#
```

```
#
# Set this property to control the intervals used in multicast
#convergance.
# The default is 3000,3000,3000,3000,3000 (all 3 seconds). The following
# example reduces the initial intervals, which may be increase responsiveness
# on fast networks:
# net.slp.multicastTimeouts=1000,1250,1500,2000,4000
# net.slp.multicastTimeouts=6000,6000,6000,6000,6000
#
# This example sets it to an aggressively short value; this may be useful
# for increased responsiveness on fast, reliable networks:
# net.slp.multicastTimeouts=1000

#
# This property works in a manner similar to net.slp.multicastTimeouts,
# but acts on DA discovery multicast convergance instead. Default is
# 2000,2000,2000,2000,3000,4000. This example increases the intervals,
# which may help to discovery DAs which are slow, heavily loaded, or on
# congested networks:
# net.slp.DADiscoveryTimeouts=3000,3000,4000,4000,4000,4000
#

#
# This property controls the retransmission intervals used for datagram
# unicast. Default is 3000,3000,3000. This example decreases the
# intervals for more aggressive retransmission, to 2 seconds:
# net.slp.datagramTimeouts=2000,2000,2000
#

#
# Set this property to change the maximum random wait bound (used with
# initial DA discovery, for instance). Default is 1000 ms (1 second).
# This example changes it to 2 seconds:
# net.slp.randomWaitBound=2000
#

#
# Change this property to change the maximum size of any datagram to
# send (MTU), in bytes. Default is 1400. This example changes it to
# 1500 bytes:
# net.slp.MTU=1500
#
```

```
#
# Set this property to restrict SLP to specific interfaces. Default is
# empty, which causes SLP to use the the default interface.
#
# net.slp.interfaces=150.132.181.22
#

#
# This property specifies the language tag to use for all messages.
# Default is "en". This example changes it to "de":
#
# net.slp.locale=de
#

#
# Set this property to limit the total number of results for any
# SLP query. Default is all results (-1). This example limits it
# to 100:
#
# net.slp.maxResults=100
#

#
# A type hint will be used if an agent needs to discover scopes with
# service agent solicitations. If the type hint is set, only service
# agents which support those types will respond. In effect, this poses
# the query: "find me all scopes which contain these service types."
#
# This sets the type hint to service:foo and service:bar:
# net.slp.typeHint=service:foo,service:bar
#
```

Ericsson Erisoft 9/S94:002 Rev E

Uppgjord - *Prepared*	Tfn - *Phone*	Datum - *Date*	Rev	Dokumentnr - *Document no*
Catarina Burman Maria Norberg		2001-08-03	A	1
Godkänd - *Approved* Ulrik Forsgren				Tillhör/referens - *File/reference*

Appendix B – Example of a register file

```
# register BasicCM
service:
BasicCmService=iiopname://172.31.24.23:12468/com/ericsson/nms/umts/ranos/BasicConf
igService_R1
scopes = default
BaseDN=Utran=Utran1
DTD=http://150.132.136.108/cds/config/cds.dtd
cimName : RanosGWKista
```

Appendix C – Example of a RANOS file

dn : cimName=RanosGWKista, cimName=Importer, cimName=APPLICATIONS, ou=CDSServer, ou=Configurations, o=Ericsson-NMS
EriNMSParam : NamingService=
EriNMSParam : BasicCmService=iiopname://172.31.24.23:12468/com/ericsson/nms/umts/ranos/BasicConfigService_R1
EriNMSParam : NotificationService=
EriNMSParam : AlarmService=
EriNMSParam : BaseDN=Utran=Utran1
EriNMSParam : DTD=http://150.132.136.108/cds/config/cds.dtd
EriNMSParam : EventSizeThreshold=3
EriNMSParam : EventTimeThreshold=60
EriNMSParam : NotificationCategories=1c1
EriNMSParam : NotificationFilter=
EriNMSParam : QueryFilter=TRUE
EriNMSParam : PollTime=30
EriNMSParam : Timeout=3
EriNMSParam : ReconnectDelay=0
cimDescription : STATUS=in contact
objectClass : top
objectClass : EriNMSApplication
cimName : RanosGWKista

Appendix D – Special information to the SLP tools

The documentation to be found over the different tools

D.1 MSLP

A protocol specification, the API specification and the service URL and service template specification.

Additional documents included in the distribution:

Building This describes how to build mslpd and libslp on the different supported platforms.

Capabilities This shows the different features that mini SLP provides against the full feature set of SLPv2.

Configuration This documents how the different components may be run and configured.

Examples This explains the two example programs included; how they may be built, run and learned from.

Unfortunately there where no examples shown.

Introduction This introduces SLP concepts.

License This reproduces the source license under which this reference i implementation is distributed.

Bugfixes and Enhancements This briefly lists the difference between Mslp1.0b1 and Mslp1.0b2.

For more information

http://www.cs.cmu.edu/~pach/mslp/documents.html

D.2 OpenSlp

Users guide consists of introduction, installation, configuration and advanced topics such as when to run slpd and security.
For more information:
http://www.openslp.org/doc/html/UsersGuide/index.html

Programmers guide describing the different functions that SLP uses.
For more information:
http://www.openslp.org/doc/html/ProgrammersGuide/index.html

RFC 2608 - Service Location Protocol, Version 2
RFC 2609 - Service Templates and Service Schemes
RFC 2610 - DHCP Options for Service Location Protocol
RFC 2614 - An API for Service Location Protocol

D.3 Solaris SLP

Administration guide consisting of for an example, an overview of SLP, how to plan the Solaris Slp deployment. It also consists of how to configure SLP properties, scopes and networkproperties and the different error messages that can occur.
For more information:
http://docs.sun.com:80/ab2/coll.47.11/SVCLOCADMIN/